

Creating Your Own Game in Excel.

Part 2

Making Your Game More Professional.

| | |
|--|----|
| 1) Introduction | 2 |
| 2) Our Guess the Number Game | 3 |
| 3) Making the Game more Professional..... | 4 |
| 4) Further Improvements to Usability | 10 |
| 5) Improving The Game. | 12 |
| 6) Part 3 of the series..... | 15 |

1) Introduction

This guide is aimed at people who are familiar with Excel, but have not programmed in it before.

During the course of this 3 part series, we will build a flexible number guessing game. Part 1 covers the planning and a very simple start to the game.

At the end of Part 1 you will have a working Guess the Number game, however it won't be the most user friendly game on the planet. The focus of Part 2 will be to improve the usability of the game, and Part 3 will focus on adding extra features to the game.

Note to more advanced users and programmers:

This is not a guide on how to program. Much of the code in this guide can be written more efficiently. This is a beginners guide on how to make a game in Excel.

Good programming practices will be covered in a later guide. The purpose for the current time is simply to build a game, which will have the secondary purpose of introducing the concept of variables and routines and other programming elements. These concepts will be later expanded upon

2) Our Guess the Number Game

In part 1 of this guide, we built a simple guess the number game. If you saved your game you can re-open it ready to carry on. If for any reason you don't have the end product from part 1, you can find the file 'creatingaGamePart1.xls in the zip file.

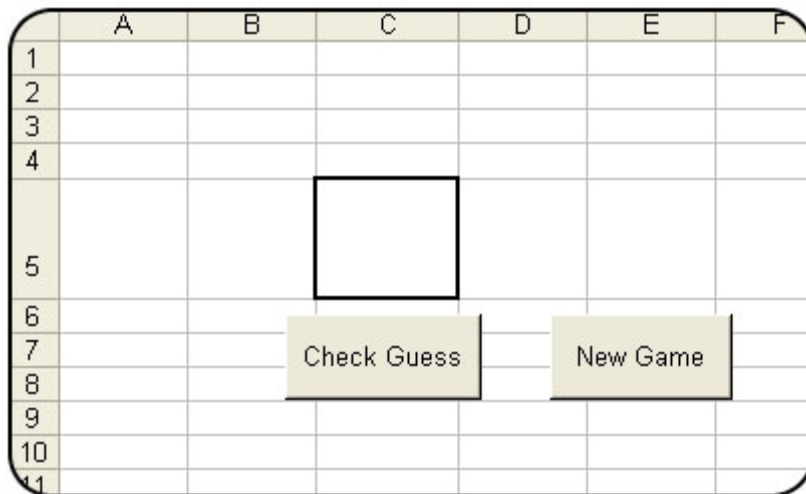
The problem with our game is that it currently doesn't explain what it's doing and doesn't explain to the player what they should be doing. This leads to a confusing experience for the player and they'll end up closing our game and never playing it again.

The purpose of this part of the guide is to tidy up the loose ends and make our game look and feel much friendlier to the player.

3) Making the Game more Professional

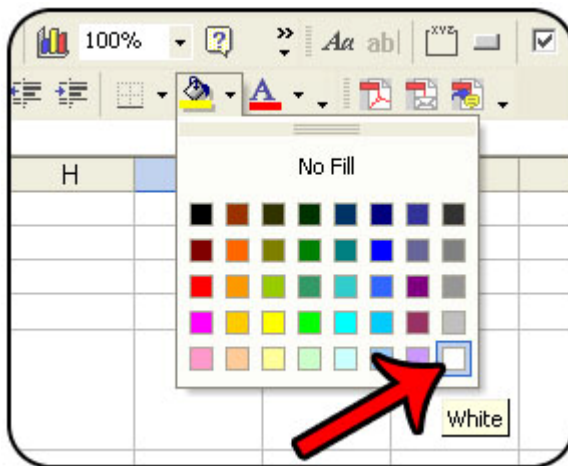
Right, do you have your game open? If not, open it now.

You should be looking at the main stage for your game, as shown below:

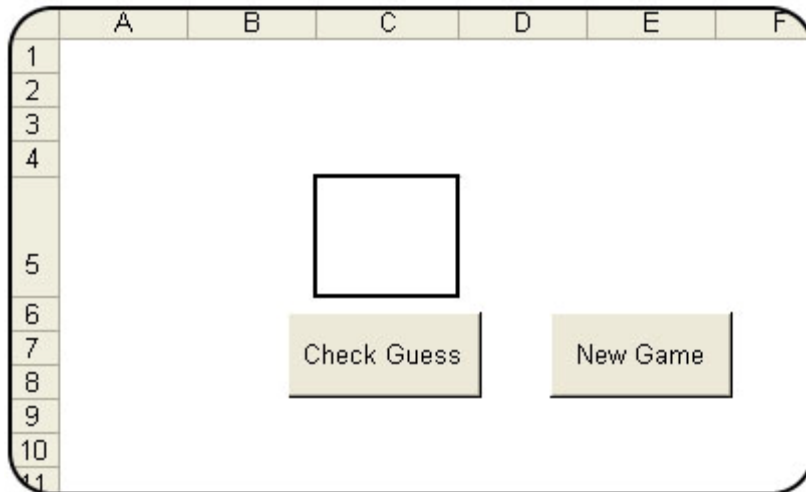


However we don't know if there is a game in progress, or if we need to start a new one. In fact, because we can see the gridlines of the cells (the light grey lines all over the place) we can't even tell what it is and what isn't part of the game.

Let's start by getting rid of the grid lines. Select all cells by left clicking on the empty square above '1' and left of 'A', and then filling the cells white.



This leaves us with a more clearly defined 'play area.'



There, that's better. It's much clearer to see what is and what isn't part of the game.

Next we need an instruction box to tell the player what they need to do.

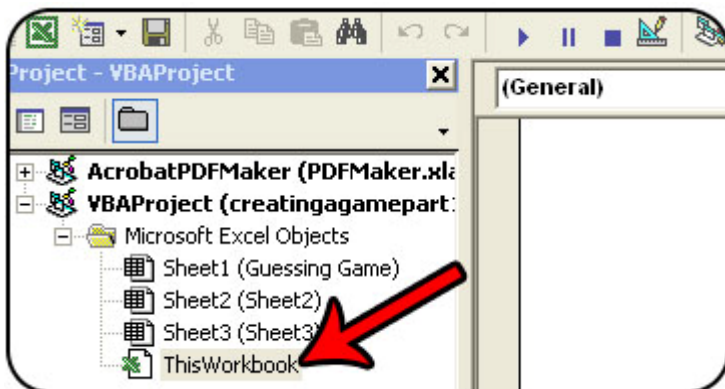
We'll use cell B2 to display our instructions, however the instructions need to be changed at different points in the game:

- | | |
|-----------------------------|--|
| Before the game has started | : Click 'New Game' to start. |
| When the game is running | : Enter a number in C5 and click 'Check Guess' |
| When the player has won | : Well done. Click 'New Game' to start. |

You might think that the first and last instructions are the same, but we need slightly different text to show that one is when a game is just not running, and another to show when a game has just finished.

Because we'll need to change the instruction as the game progresses, we will need to make the changes by code.

Open up your VBA editor (ALT+F11) and double click on the 'ThisWorkbook' item in the project window.



This will open up a new sheet in which to write code.

Now is a good time to talk about events.

Events occur whenever something changes, and we can use events to start doing things. I know that doesn't make sense now but we'll use a real world example, and then tie it up to your code.

Image you're at home. The doorbell rings and you go answer the door. You have just responded to an event. The event was `doorbell.ring`, and your response was to go and answer the door. You responded to an event by taking action.

If we could write a macro for this it might look a little like the following:

```
private sub doorbell.ring()  
  
    getup off lazy butt  
    goto.door  
    open.door  
    greet.unwelcomeVisitor  
  
End sub
```

An event occurred and you took action in response to that event.
Now we'll translate that into our game:

Your 'New Game' button is the doorbell. Clicking that button makes it ring. In response to the ringing, Excel performs some actions.

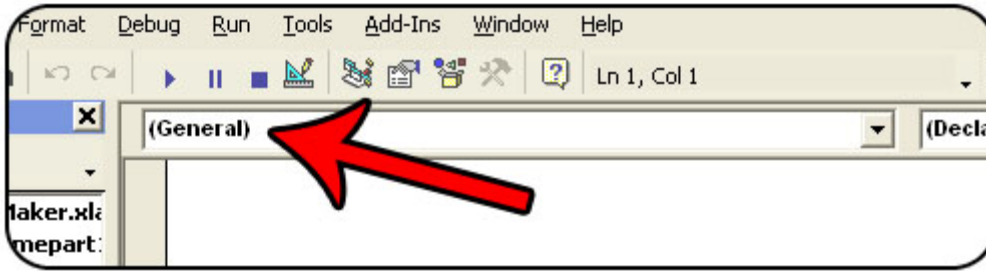
In Excel however, instead of actually ringing, the button exclaims that it's been clicked. It sort of shouts Hey, Excel, the player just clicked me. Excel registers that a click event has just occurred on one of its buttons.

Excel then checks to see if any macros are assigned to the buttons' Click event, and sees there is one (remember when we assigned macros back in Part 1?). If there is a macro assigned to the buttons' Click event, Excel runs that macro.

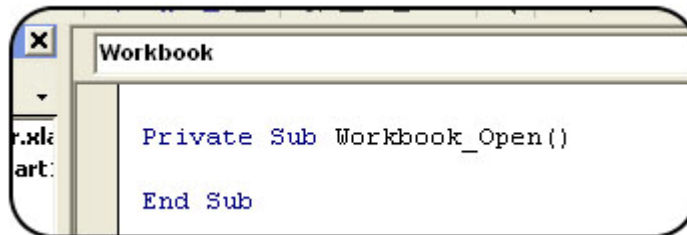
It's not just buttons that have events. For example, this whole workbook has events. When you open your game, the `workbook_Open` event is launched. There are many other events, such as `workbook_Save`, and `workbook_Close` that we can use, but for the time being, we want the `workbook_Open` event.

When Excel detects an Open event for this workbook, it will run code in the `workbook_Open` macro. At the moment, this macro doesn't exist, so nothing special happens when you open the workbook.

Let's start writing our `workbook_Open` macro.
In the VBA editor, use the Object drop down list:



Then select 'Workbook' from the list.
This will create the workbook.Open macro, as shown below:



Inside of our macro, write the following code:

Note:

The following code should all be on a single line. The width of the page here makes the code spill over to two lines.

```
response = msgbox("Welcome to Guess  
the Number!", vbokonly, "Welcome")
```

```
Private Sub Workbook_Open()  
    response = MsgBox("Welcome to Guess the Number!", vbOKOnly, "Welcome")  
End Sub
```

Now close the VBA editor, save Excel and then close Excel.
Now open your file back up and voila! it should greet you upon opening.

That is an example of Excel catching an Open event, and responding with a macro.

There is more that we want to do when the game first loads up. We first want to make sure the player can see the main playing area. If the player moves to a different sheet (like sheet3) and then saves the file and closes it, when they open it again, it'll open on sheet 3. We need to force the game to show sheet 1 each time just in case.

I know we could just delete the unwanted sheets from the workbook, and then the player wouldn't be able to move around, but in practice your games will nearly always have more than just one sheet, so we need to keep the player in check and force things to happen every so often.

We'll do this by jumping to the right sheet, and by selecting cell A1. (maybe the last player hit page-down a few times, so by selecting A1 we are getting back to the top left hand corner of the sheet).

Write the following code into your workbook_Open macro **before** the msgbox line:

```
sheets("Guessing Game").select  
range("A1").select
```

Note:

If you called your sheet something else, you will have to ensure you type the correct name into the code.

While we're here, we should make sure cell A1 is empty. If it still contains a number from a previous game, the player might spot that and realize cell A1 contains the hidden number.

add the following line of code:

```
range("A1").value = ""
```

```
Private Sub Workbook_Open()  
  
    Sheets("Guessing Game").Select  
    Range("A1").Select  
    Range("A1").Value = ""  
  
    response = MsgBox("Welcome to Guess the Number!", vbOKOnly, "Welcome")  
  
End Sub
```

So now when the game opens, it'll jump to the right sheet, and make sure it's looking at the right part of that sheet before displaying a welcome message.

Things are looking better.

Once the player clicks ok to dismiss the welcome message, we need to write the first instruction.

Add the following code to your macro:

```
range("B2").value = "click the 'New Game' button."
```



```
Private Sub Workbook_Open()  
  
    Sheets("Guessing Game").Select  
    Range("A1").Select  
    Range("A1").Value = ""  
  
    response = MsgBox("Welcome to Guess the Number!", vbOKOnly, "Welcome")  
  
    Range("B2").Value = "Click the 'New Game' Button."  
  
End Sub
```

So that's the first part of the game tidied up.

The game looks tidier and is more clearly defined as to what is and what is not part of the game.

It and it gives instructions on what the player needs to do, however, If you look in Excel at cell B2, it does not say Click the 'New Game' button. This is because we have placed the code to do that in the workbooks Open event. That code will only run when the workbook is opened.

To test this out, save and close your game, and then open it again. You should get a message box welcoming you, and then the instruction will appear in cell B2.

The instructions are a little small, so go ahead and change the font size in cell B2 to something a larger, like 16.

That makes the start of the game much more user friendly.

We can now better identify the game area, we are greeted when we open the game and we have instructions on what to do next.

The next section will walk you through some steps to make playing the game more user friendly.

4) Further Improvements to Usability

So we now have a game that greets the player, and instructs him or her on how to start a new game. At this point, the game stops being friendly. Let me give you an example of this:

You have opened the game, and it is telling you to click on the 'New Game' button to begin.

So, we click on the 'New Game' button..... and nothing happens!

Well, in truth, Excel has picked a random number between 1 and 10 and placed that number in cell A1 and is now waiting for us to see if we can guess that number, but we can't expect the player to know this, so we have to go back into the code and make a few changes.

Jump back to the VBA editor, and double click on the 'Sheet1 (Guessing Game)' entry in Project explorer (top left).

This will open up the code in this sheet. We have two macros here already:

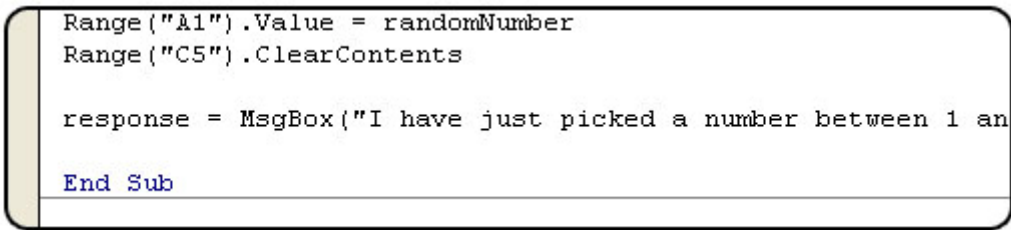
```
startGame  
checkNumbersForMatch
```

The start game macro picks a random number and puts it into cell A1, so that is the macro we want to edit.

After the line that reads `range("C5").clearContents`, add the following code:

Note: this code should be written all on one line. It will probably fall off of the right hand side of the screen. That is ok.

```
response = msgbox("I have just picked a number between 1 and  
10. See you if can guess that number in the box on the  
screen.", vbokonly, "Game Started")
```



```
Range("A1").Value = randomNumber  
Range("C5").ClearContents  
  
response = MsgBox("I have just picked a number between 1 an  
End Sub
```

Once the game has started, we want Excel to automatically select cell C5, ready for our player to enter a guess, so enter the following line below the previous:

```
range("C5").select
```

```
Range("C5").ClearContents  
  
response = MsgBox("I have just picked a number between 1 and  
Range("C5").Select  
  
End Sub
```

Now return to Excel and click on the 'New Game' button.

When the player clicks on the 'New Game' button, he/she can be sure that a new game has started, and that they know what to do.

Now that we have started a game, and are trying to guess the number, we don't want our instruction in cell B2 to tell the player to start a new game. We can change this here by adding the following line of code:

Note: This code should all be on one line in the VBA editor.

```
range("B2").value = "Enter your guess in the box and click  
'Check Guess' "
```

```
response = MsgBox("I have just picked a number between 1 and 10. See you in  
Range("C5").Select  
  
Range("B2").Value = "Enter your guess in the box and click 'Check Guess' "  
  
End Sub
```

5) Improving The Game.

All the improvements we have made so far in this part have been to increase the usability of the game, to make it more user-friendly.

Now we will add a new feature to the game. Rather than Excel just telling us that our guess is wrong, we will have it respond in a Higher or Lower manner, allowing us to refine our guesses, rather than just go 1, 2, 3, 4, 5, 6, 7, etc until we hit the right number.

To do this, we need to go into our `checkNumberForMatch` macro. In the VBA editor, scroll down to this macro and find the line that starts reads :

```
If hiddenNumber = playerGuess Then
```

This is the code that checks to see if you've guessed the right number or not.

The next line of code runs if you have guessed the right number, or the line underneath `Else` runs if you have not guessed the correct number.

Instead of just displaying a message, we want to tell the player to either guess higher, or guess lower next time. This means writing another IF statement.

At the moment the code is structured like this:

```
If yourGuess = hiddenNumber then
    message("You Win")
Else
    message("You Lose")
End If
```

Instead of having the message "You Lose", we can add another If statement, which I'll colour differently here to highlight the two separate If statements.

```
If yourGuess = hiddenNumber then
    message("You Win")
Else
    If yourGuess is less than hiddenNumber then
        message("Higher!")
    Else
        message("Lower!")
    End If
End If
```

In the example above, If your guess is correct, you will get a "You Win" message, and that's it. then code will then jump down to the black `End If`.

If however your guess is wrong, Excel will then check to see if your guess is lower than the hidden number, and if it is, it will display "Higher!", telling the player they need to guess higher.

If your guess is not lower than the hidden number, then you must have guessed too high, so Excel will display “Lower!” telling the player they need to guess lower.

Lets write this code into your game.

You will need to delete the line of code that displays a message box telling the player they were wrong. The picture below highlights the line you need to delete.

```
If hiddenNumber = playerGuess Then
    response = MsgBox("Well done. you guessed the right number")
Else
    response = MsgBox("I'm sorry, that's not the right number")
End If
```

Now replace it with the following code:

```
If playerGuess < hiddenNumber then
    response = MsgBox("Higher!")
Else
    response = MsgBox("Lower!")
End If
```

So your code should now look as follows:

```
If hiddenNumber = playerGuess Then
    response = MsgBox("Well done. you guessed the right number")
Else
    If playerGuess < hiddenNumber Then
        response = MsgBox("Higher!")
    Else
        response = MsgBox("Lower!")
    End If
End If
```

Now try the game out. Excel should give you instructions on whether you need to be guessing higher or lower than your previous guess.

This makes it a little more interesting than just trying all numbers between 0 and 10.

Finally, we will change the instructions in cell B2 to tell use we’ve finished the game, and how to start a new game.

We have to find the piece of code the check to see if we’ve guessed the right number, and then display a message box telling us this. See you have can find this section of code yourself, but don’t worry if you can’t just yet. I’ll show you below.

If you didn't manage to find it, look for the code that reads:

```
If hiddenNumber = playerGuess Then
```

This line of code checks the hidden number to see if it matches your guess. If it does, the following line of code is run:

```
response = MsgBox("Well done. you guessed the right number",  
vbOKOnly, "You Win")
```

This line of code displays a message box. If you take a look at your code in the game, this is the last thing that happens when you guess the number correctly. The rest of the code is run if your guess is **NOT** correct. This is due to the way IF statements are structured.

I've already explained IF structures, but not in great detail, so here is another explanation of how they work.

Note: I've included line numbers so I can explain things a little easier afterwards.

```
1      IF this tutorial is good THEN  
2          Do this line of code  
3          and this line  
4          and as many other lines as we need  
5      OTHERWISE  
6          Do this line instead, not those above here  
7          and do this line  
8          and also do this line  
9      END IF  
10     continue on with rest of macro
```

First line 1 gets processed. We do a check to see if this tutorial is good.

If it **IS** good, lines 2, 3, and 4 get processed, then we jump down to line 9 and 10.

If however, the tutorial is not good, lines 2, 3, and 4 get skipped and lines 6, 7, and 8 are run, followed by 9 and 10.

So, if we want to add code that change our instructions after we have finished the game, we need to put it with the code the tells us we've finished our. Add the following line of code above the first line of code that reads "Else".

```
range("B2").value = "Well done. Click 'New Game' to start."
```

Your code should now look like the following diagram:

```
If hiddenNumber = playerGuess Then
    response = MsgBox("Well done. you guessed the right number",
    Range("B2").Value = "Well done. Click 'New Game' to start."
Else
    If playerGuess < hiddenNumber Then
        response = MsgBox("Higher!")
    Else
        response = MsgBox("Lower!")
    End If
End If
```

That's it for part 2 of this guide. Save your work and give the game a try. I'm sure you'll agree that your game is far better now than it was at the end of part 1, where you had no instructions, no feedback and a messy play area.

6) Part 3 of the series.

In part 3, we will add a few more features to our game to make it more flexible. This will involve changes both on the game stage, and in the code, and will give our simple little game more longevity by allowing the player to customize the game to their liking.

We will add features that allow the player to:

- Change the range of numbers (0-10). Player could pick their own ranges, such as 0-20, or 12-40, or 1,000-10,000.
- Count the number of guesses it takes to find the hidden number.
- Set a number of tries they are allowed, such as 3 guesses. If the hidden number is not found within 3 guesses the player loses. For larger ranges (0-1000), the player will need to increase the number of allowed guesses.